Zadanie č.2C: Genetický algoritmus a Zakázané prehľadávanie*

Hlib Kokin ID: 117991
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xkokin@stuba.sk

06.11.2022

*Dokumentacia druheho zadania v predmete Umelá Inteligencia, ak. rok 2022/23, vedenie: Ivan Kapustík

Abstrakt

Dokumentácia k tejto úlohe obsahuje podmienky samotnej úlohy, popis použitých algoritmov a ich vlastností, analýzu zdrojového kódu, testy a grafy s konkrétnymi výsledkami, na základe ktorých je možné posúdiť efektívnosť genetického algoritmu a algoritmu Zakázané prehľadávanie (Tabu Search).

Obsah

Dokumentaciá zadania č.2

1 Zadanie

Podľa podmienok zadania som mal použiť dva rôzne algoritmy (Genetické a Zakázané vyhľadávanie) na nájdenie najkratšej cesty medzi mestami (20 - 40 miest) na mape s rozmermi 200 x 200 km. Cesta musí byť vo forme sledu miest

2 Rozbehanie Projektu

Svoju úlohu som sa rozhodol implementovať v jazyku Python, vo vývojovom prostredí PyCharm, na spustenie main funkcie je potrebné nainštalovať balík MatPlotLib verzie 3.5.3, aby ste mohli zobraziť grafy novo vygenerovaných riešení

3 Genetický algoritmus

Genetický algoritmus funguje na základe výberu najlepšieho génu z generácie a jeho miešania s inými, menej úspešnými génmi pomocou mutácií a crossoverov, kým nie sú splnené podmienky alebo sa nedosiahne určitá generácia.

3.1 Reprezentácia génov

Jeden gén predstavuje zoznam miest v určitom poradí, gény sa vytvárajú náhodným usporiadaním počiatočného zoznamu miest, kde prvé mesto vždy zostáva prvým a všetky ostatné sú náhodne premiešané pomocou funkcie sample zabudovanej v Pythone v balík random.

Jedna generácia pozostáva z permutácie 10 rôznych génov, ktorých obsah je v čase inicializácie náhodne vybraný na základe zoznamu vybraných miest.

3.2 Inicializacia prvej generácie

Aby sme získali našu prvú generáciu, vezmeme si zoznam našich vybraných miest a generáciu naplníme 10-krát volaním funkcie regenerácie génov, čím dostaneme prvú generáciu pozostávajúcu z 10 rôznych génov.

3.3 Spôsob tvorby novej generácie

Aby sme mali dostatok generácií na hľadanie riešenia, zadáme slučku pre (počet miest * 200) opakovaní. (V prípade 20 miest to bude 4000 opakovaní).

V každej iterácii tejto slučky zoradíme predtým prijatú alebo upravenú generáciu podľa kritéria vhodnosti (fitness), takže na prvom mieste bude vždy cesta s najkratšou vzdialenosťou

V prvom rade rozdelíme prvých 15 percent súčasnej generácie na ďalšiu, takzvané elitné gény. Potom v cykle na dĺžku génu - počet elitných génov opakovaní budeme robiť crossovery a mutácie, aby sme sa pokúsili nájsť lepšie riešenie, než aké máme v súčasnosti.

V prvom type selekcie sme ako prví pre crossover vezmeme prvý elitný gén a jeden gén zo zvyšných zo súčasnej generácie, po dokončení crossoveru je šanca, že mutácia bude aplikovaná aj na zmenený gén a pridame ho do novej generácie, V druhom type selekcie tiež skrížime dvoch rodičov v crossovere, zmutujeme

výsledok kríženia a potom porovnáme výslednok mutacie s mutaciou toho, ktorý bol použitý ako jeho druhý rodič (prvý rodič je najlepší elitný gén) a ak mutacia rodiča zostane lepša, tak pridá sa do novej generácie, a výsledok crossoveru nie. Podľa výsledkov testov sa zistilo, že genetický algoritmus s selekciou druhého typu nachádza riešenie horšie ako selekcia prvého typu a často na mladších generáciách, je to spôsobené tým, že druhý typ si vždy vyberie najlepší výsledok medzi mutovaným rodičom a skríženými rodičmi, hoci najhorší z týchto dvoch má potenciál môže byť viac (s najväčšou pravdepodobnosťou preto prvý typ selekcie nájde najlepšie riešenie na vyšších generáciách, zatiaľ čo druhý typ nájde riešenie na nižších generáciách, ale častejšie horšie)

3.4 Cycle Crossover

Na začiatku funkcie napíšeme dva zoznamy s postupnosťou mestských čísel pre gény, ktoré sú rodičmi nového génu. Potom budeme postupovať nasledovne, napríklad máme dva gény:

$$P_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8),$$

$$P_2 = (8\ 5\ 2\ 1\ 3\ 6\ 4\ 7).$$

Obr. 1: Cycle Crossover Rodičia

Teraz je už len na nás, aký prvý kúsok pre potomka vyberieme buď od prvého alebo od druhého rodiča. V našom príklade musí byť prvý bit potomka 1 alebo 8. Zvoľme, aby bol 1. Teraz by mal byť každý bit potomka odobratý od jedného z jeho rodičov s rovnakou pozíciou, to znamená, že ďalej už nemáme na výber, takže ďalší bit, ktorý treba zvážiť, musí byť bit 8, keďže bit od druhého rodiča je tesne pod vybraným bitom 1. V prvom rodičovi je tento bit na 8. pozícii; teda

$$O_1 = (1\ \times\ \times\ \times\ \times\ \times\ \times\ 8).$$

Obr. 2: Cycle Crossover Postup (1)

Táto účasť zahŕňa bit 7, čo je bit druhého rodiča tesne pod vybraným bitom na 7. pozícii prvého rodiča. Teda Ďalej nás to prinútilo umiestniť 4 na 4. miesto.

$$O_1 = (1\ \times\ \times\ 4\ \times\ \times\ 7\ 8).$$

Obr. 3: Cycle Crossover Postup (2)

Potom príde 1, ktorá je už v zozname; tak sme dokončili cyklus a vyplnili zostávajúce prázdne pozície bitmi tých pozícií, ktoré sú v druhom rodičovi:

Koncept tohto algoritmu bol implementovaný nasledovne:

$$O_1 = (1\ 5\ 2\ 4\ 3\ 6\ 7\ 8).$$

Obr. 4: Cycle Crossover Postup (3)

Po napísaní dvoch zoznamov s číslami miest začneme cyklus (počet opakovaní - dĺžka génu), v ktorom sa budeme pohybovať cez rôzne indexy zoznamu, začneme od 0. Najprv doplníme aktuálne číslo mesta do zoznam navštívených miest, ako aj index k zoznamu použitých indexov, aby ste dvakrát netrafili ten istý bod. Potom, ak sme ešte nedokončili cyklus vypĺňania nového génu bodkami od prvého rodiča, tak k novému génu pridáme mesto (od prvého rodiča A) s indexom, ktorý sme našli a nový index zapíšeme do premennej, ktorý sa bude rovnať číslu mesta z génu rodiča B (v tom istom indexe, pri ktorom sme prevzali prvok z rodiča A), potom skontrolujte, či sme tento index už predtým použili, ak áno, nastavime hodnotu indexa k prvému bodu, ktorý ešte nemáme naplneny a začneme vypĺňať prazdne miesta v novom géne tými mestami, ktoré sú na rovnakých miestach od rodiča B.

3.5 Mutacia

Mutácia bola realizovaná nahradením dvoch náhodne vybraných miest, prvé vyberieme náhodne a druhé tak, aby sa nerovnalo prvému, potom mutujeme s možnosťou 50 percent.

4 Zakázané prehľadávanie

Zakázané vyhľadávanie je algoritmus, ktorý vyberá najlepšie riešenie v každom kroku algoritmu, pričom preskakuje zakázané kroky, aby sa zabránilo slučke v lokálnom maxime. Po dosiahnutí maximálnej veľkosti zoznamu zakázaných štátov sa z neho odstráni prvý prvok.

4.1 Postup algoritmu

Pri implementácii tohto algoritmu je prvým krokom zoradiť zoznam miest tak, aby vzdialenosť od prvého mesta k druhému (podľa indexu) bola čo najkratšia, potom prvé mesto pridáme do zoznamu zakázaných miest (štátov) a spustíme cyklus, v ktorom sa dostaneme na koniec zoznamu miest, zmeníme ich poradie a získame najrýchlejšie riešenie.

V každej iterácii cyklu vezmeme aktuálne mesto a skontrolujeme vzdialenosť ku každému zo zostávajúcich miest (okrem tých, ktoré sme zakázali, 30 - 50 percent všetkých miest) a zapíšeme najlepšiu zistenú vzdialenosť do premennej, aby sme potom mohli porovnávať novo vypočítanú vzdialenosť s najlepšou. Keď sa nájde najlepšie mesto, pridáme ho do zoznamu zakázaných miest, skontrolujeme, či je zoznam plný (ak je zoznam plný, odstránime odtiaľ 2. prvok, pretože prvý prvok je počiatočné mesto) a vrátime sa naše najlepšie mesto

Potom môžeme nájdenú najlepšiu možnosť umiestniť na miesto nasledujúceho mesta za aktuálnym bodom v zozname miest (náhrada) Keď sa cyklus

skončí, na koniec zoznamu pridáme naše počiatočné mesto a vrátime poradie miest.

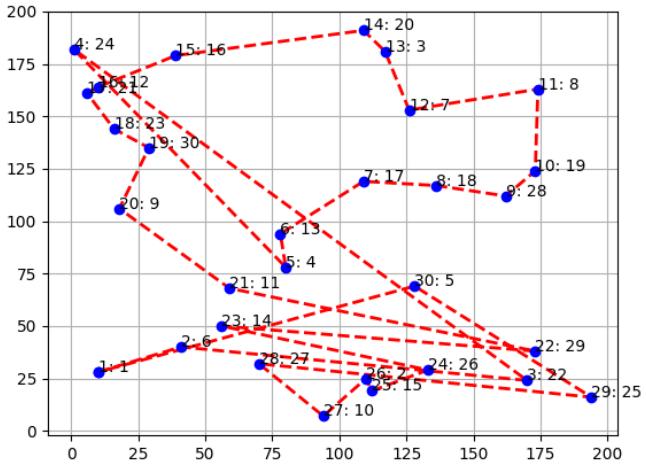
4.2 Zakázané stavy

Blokované stavy v tomto probléme sú mestá, ktoré sme navštívili v posledných krokoch, napríklad v prípade 21 mest (a rozsahom tabu zoznamu 30%) bude v zozname blokovaných miest maximálne 7 miest. V čase pridania 8. mesta - prvý index zoznamu bude vymazaný.

V procese testovania na rôzne generovaných mestách sa zistila vlastnosť, že čím bližšie sú mestá k stredu a čým menší je rozdiel vo vzdialenosti medzi nimi, tým väčší musí byť zoznam zakázaných stavov aby nedošlo k cyklu v lokalnom maxime.

Podľa výsledkov testov sa ukázalo, že ak by sa mestá generovali bližšie k okraju mapy, tak by stačil zoznam blokovaných štátov pre 10 - 30 percent z počtu miest, ale ak by sa mestá generovali bližšie do centra a vzdialenosť medzi nimi je blizko rovnaká, potom zoznam asi 50 percent z celkového počtu miest. Ale aj so zoznamom polovice miest z ich celkového počtu bude táto metóda fungovať rýchlejšie ako kontrola BruteForce.

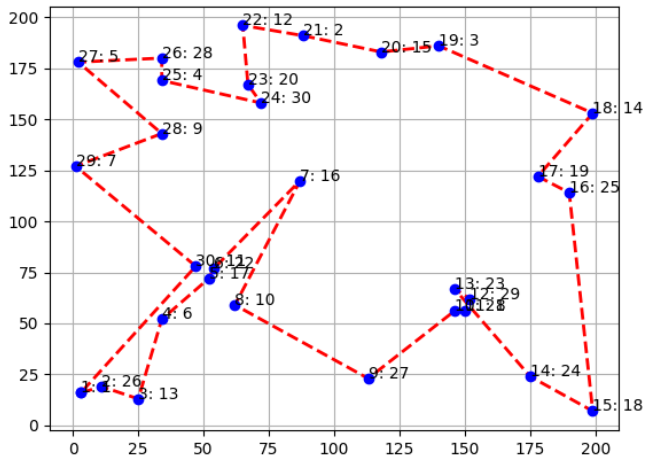
Tu je príklad, keď bol algoritmus zakázaných vyhľadávaní testovaný pre 30 miest so zakázaným zoznamom s veľkosťou 15 miest (50 percent), dokonca aj s takým veľkým zoznamom zakázaných miest v určitom okamihu došlo k cyklu a nedostali sme to najlepšie. výsledkom tohto usporiadania miest.



Obr. 5: Cyklenie v lokalnom maxime

A tu s rovnakými parametrami, ale inak vygenerovanými mestami už máme prijateľný výsledok cesty. Dá sa usúdiť, že umiestnenie miest na mape výrazne ovplyvňuje výkon a veľkosť zoznamu blokovaných stavov

Aby sa dosiahla stabilita výsledkov, bolo rozhodnuté nastaviť veľkosť zoznamu blokovaných štátov na 35 percent z celkového počtu miest pre testy 20 miest a 50 percent pre test 30 miest.



Obr. 6: Bez Cyklenia

5 Testovacia funkcia

5.1 Generácia mest

Mestá sa generujú hneď na začiatku, počet miest odovzdáme funkcii generate_cities, ktorá zasa zo zoznamu možných miest vyberie 20 miest bez opakovania (200 * 200 = 40 000 možností) pomocou funkcie sample balenie random.

5.2 Kreslenie grafov

Výsledok náhodného generovania odovzdávame nasledujúcim funkciám run_evolution(genetický algoritmus) a run_tabu_search(zakazane prehľadavanie), každá nám vráti postupnosť miest – najlepšie riešenie, aké funkcia mohla nájsť. Funkcia genetického algoritmu sa volá 6-krát, potom sa z výsledkov vyberie tá najúčinnejšia. Pre prvé 3 iterácie funguje genetický algoritmus v štandardnom type výberu (Najskôr skríži dvoch rodičov a potom mutuje výsledok) a druhý - v inom type (Výsledok je skrížený a mutovaný, ale porovnáva sa s mutáciou druhého rodiča a vyberie sa ten najlepší) Podľa výsledkov funkcií algoritmov zostavíme grafy, podľa mapy cesty pre každý algoritmus a podľa grafu fitnessu pre geneticky algoritmus (štyri grafy pre geneticky algoritmus: prvy typ - 2 a druhy typ - 2) spolu budeme mať 10 grafov, pretože testy priebehaju 2-krat, prvykrat pre 20 mest a druhy - pre 30 mest.

Prvý, tretí, šiesty a ôsmy graf zobrazuje cestu nájdenú genetickým algoritmom, prvý a tretí - pre 20 miest s prvým a druhým typom výberu a šiesty a ôsmy - pre 30 miest s prvým a druhým typom výberu, resp. Grafy, ktoré prichádzajú bezprostredne po grafoch s cestou nájdenou genetickým algoritmom (stĺpce 2, 4, 7, 9), sú grafy znázorňujúce progresiu fitness zložky (každe 300 generacie) najlepšieho génu v generácii. Grafy v poradí 5 a 10 sú grafy zobrazujúce riešenia nájdené algoritmom Tabu Search pre 20 a 30 miest.

5.3 Zaznamenávanie testovacích časov

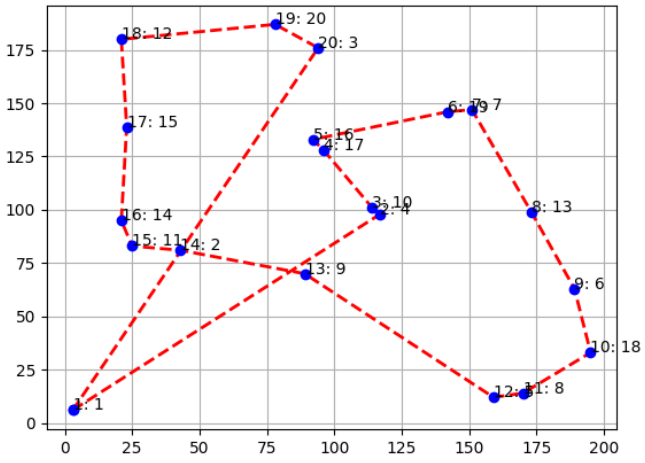
Čas testu zoberiem klasickým spôsobom, zavolám funkciu, ktorá vráti aktuálny čas a po vykonaní potrebnej časti kódu odčítam začiatočny čas od bežneho času. Čas na hľadanie riešenia v genetickom algoritme si zapíšem v momente, keď sa nájde najlepšia cesta zo všetkých predchádzajúcich, ak neskôr nájdem lepšie riešenie, tak sa čas prepíše.

6 Výsledky testov

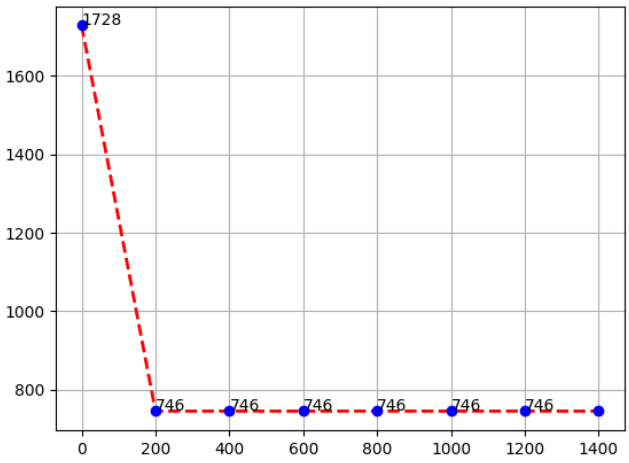
6.1 20 Mest

Nižšie sú uvedené grafy riešenia nájdeného genetickým algoritmom pre náhodne vygenerovaných 20 miest a prvý graf ukazuje cestu cez týchto 20 miest so symbolmi nad bodmi, kde prvé číslo je poradie navštevujúcich miest a druhé číslo je poradové číslo mesta počas generovania.

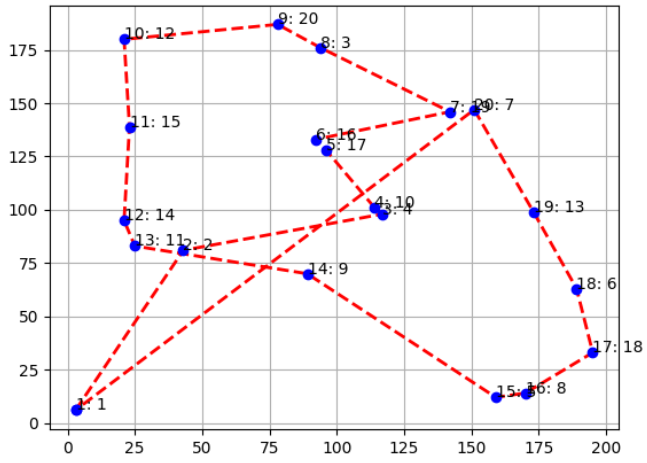
Prvý graf ukazuje riešenie nájdené genetickým algoritmom s prvým typom selekcie a tretí graf ukazuje riešenie nájdené algoritmom s druhým typom selekcie.



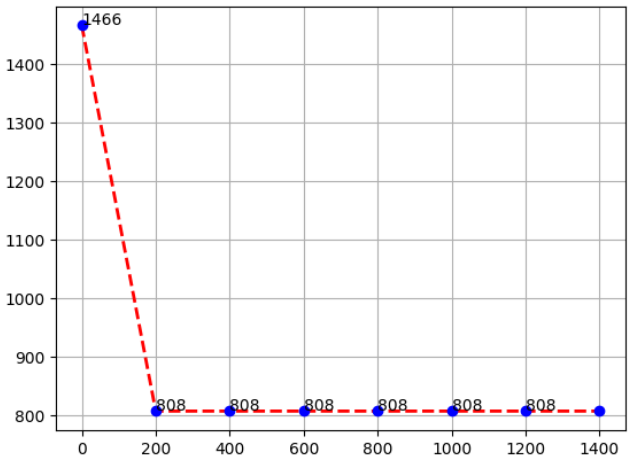
Obr. 7: Geneticky algoritmus prveho typu selekcie na 20 mest, mapa riešenia



Obr. 8: Progresie fitness parametra najlepšieho génu každých 500 generácií

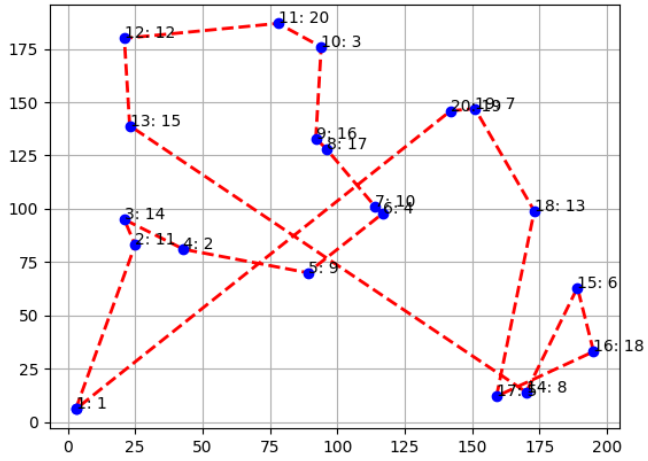


Obr. 9: Geneticky algoritmus 2ho typu selekcie na 20 mest, mapa riešenia



Obr. 10: Progresie fitness parametra najlepšieho génu každých 500 generácií (2)

Nižšie je uvedeny grafy riešenia nájdeného algoritmom zakázaného vyhľadávania s veľkosťou zoznamu zakázaných miest 35 percent z celkového počtu miest.



Obr. 11: Zakazane Prehľadavanie na 20 mest, mapa riešenia

Vzdialenosť zistená genetickým algoritmom je o 135, 62 km (939km a 1012 km) kratšia ako vzdialenosť zistená algoritmom zakázaného vyhľadávania (1074 km). Čas genetického algoritmu prveho typu selekcie je 0,20 sekundy, riešenie bolo nájdené v 458 generácie z 4000 generácií. Čas genetického algoritmu druheho typu selekcie je 0,30 sekundy, riešenie bolo nájdené v 408 generácie z 4000

generácií. Čas algoritmu zakázaného vyhľadávania je 0.10 sekundy.

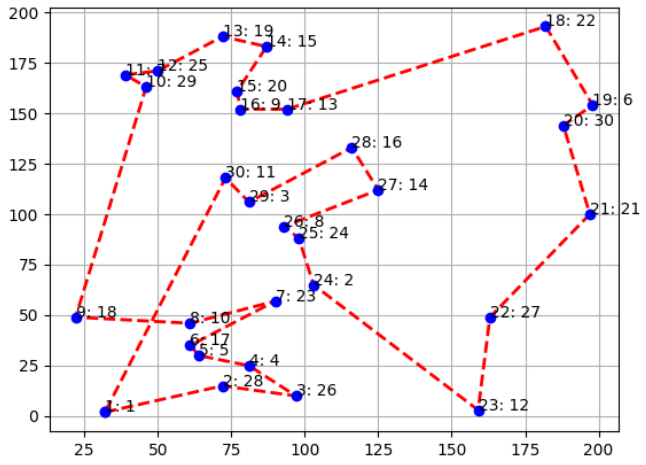
Z tohto testu, ako aj testov, ktoré neboli zahrnuté v dokumentácii, vyplynulo, že genetický algoritmus prvého typu selekcie nájde riešenie rýchlejšie a lepšie ako druhý typ selekčného algoritmu, hoci druhý typ selekcie selektuje najlepšie gény na základe fitness kondície.

Hoci rozdiel v čase hľadania riešenia v tomto teste nie je taký závažný medzi genetickým algoritmom a zakázaným vyhľadávacím algoritmom, v iných testoch, kde boli riešenia genetického algoritmu na starších generáciách, sa zakázaný vyhľadávací algoritmus vyrovnal rôznymi spôsobmi rýchlejšie, aj keď trochu horšie (ako sme už zistili - v algoritme zakázaných vyhľadávaní hrá veľkú rolu poloha miest a teda aj veľkosť zoznamu zakázaných štátov... Pri určitom umiestnení miest 35 percent celkový počet miest jednoducho nemusí stačiť, takže výsledok nebude najlepší.)

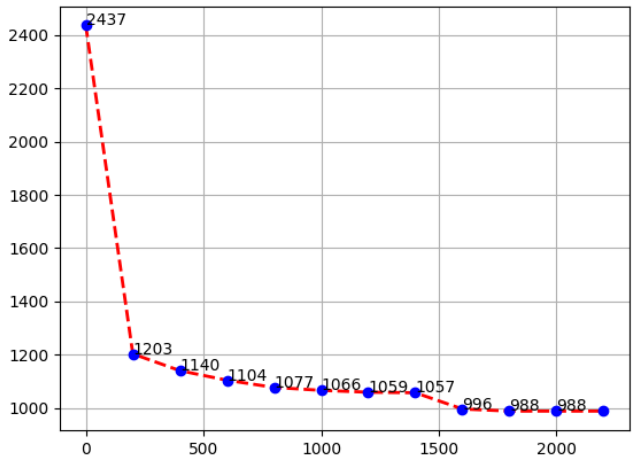
6.2 30 Mest

Nižšie sú uvedené grafy riešenia nájdeného genetickým algoritmom pre náhodne vygenerovaných 20 miest a prvý graf ukazuje cestu cez týchto 20 miest so symbolmi nad bodmi, kde prvé číslo je poradie navštevujúcich miest a druhé číslo je poradové číslo mesta počas generovania.

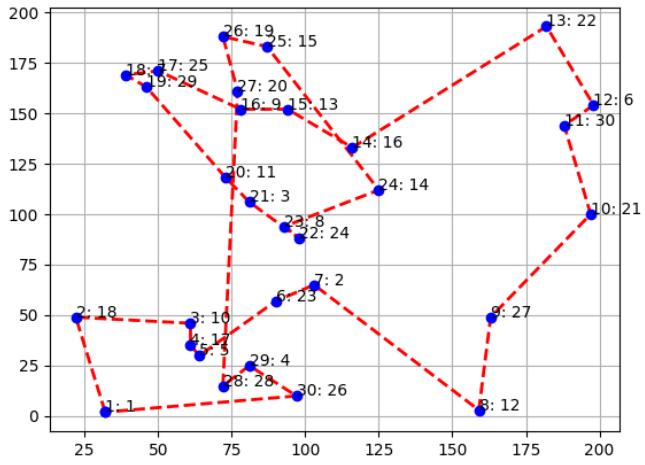
Prvý graf ukazuje riešenie nájdené genetickým algoritmom s prvým typom selekcie a tretí graf ukazuje riešenie nájdené algoritmom s druhým typom selekcie.



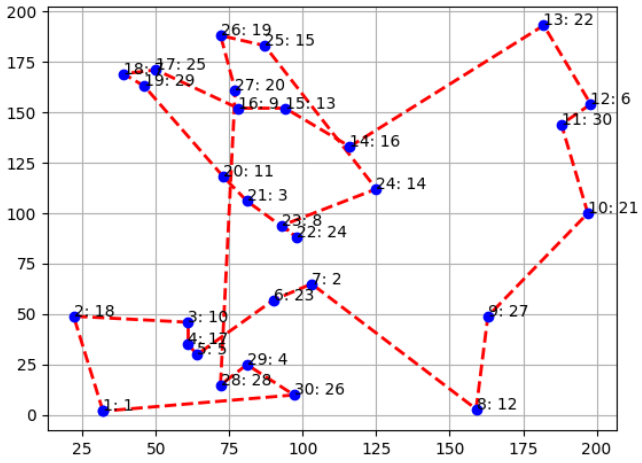
Obr. 12: Geneticky algoritmus prveho typu selekcie na 30 mest, mapa riešenia



Obr. 13: Progresie fitness parametra najlepšieho génu každých 500 generácií

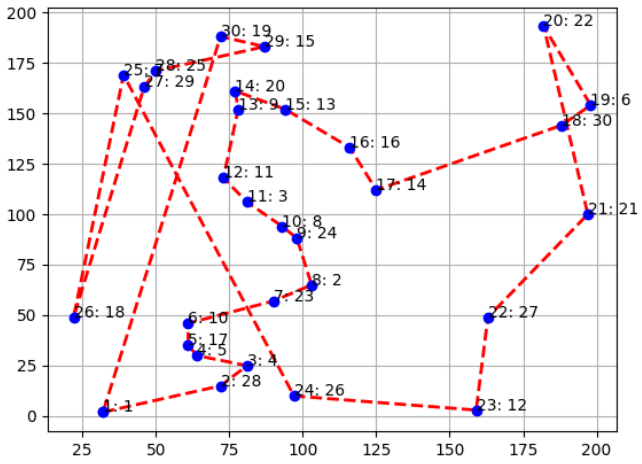


Obr. 14: Geneticky algoritmus 2ho typu selekcie na 30 mest, mapa riešenia



Obr. 15: Progresie fitness parametra najlepšieho génu každých 500 generácií (2)

Nižšie je uvedeny grafy riešenia nájdeného algoritmom zakázaného vyhľadávania s veľkosťou zoznamu zakázaných miest 50 percent z celkového počtu miest.



Obr. 16: Zakazane Prehľadavanie na 30 mest, mapa riešenia

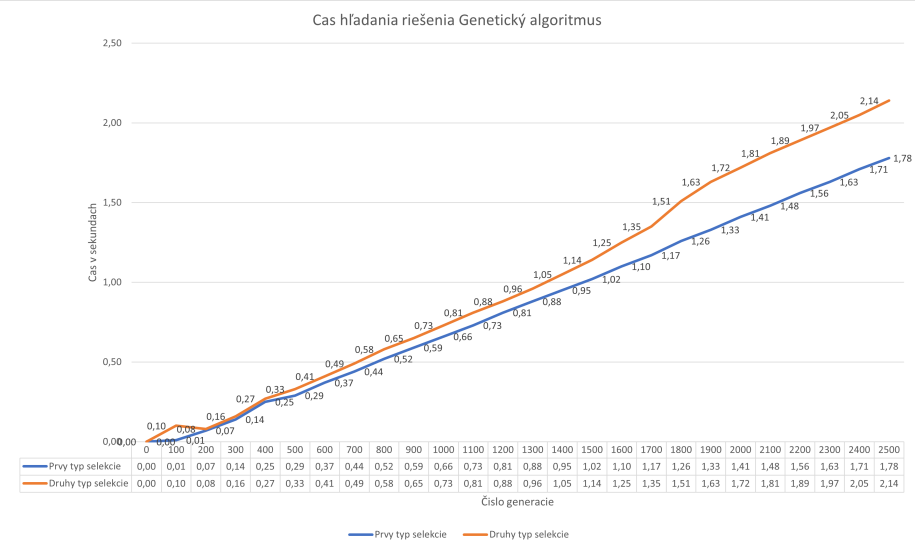
Vzdialenosť zistená genetickým algoritmom je o 254, 191 km (1111km a 1144 km) kratšia ako vzdialenosť zistená algoritmom zakázaného vyhľadávania (1365 km). Čas genetického algoritmu prveho typu selekcie je 2.84 sekundy, riešenie bolo nájdené v 4390 generácie z 6000 generácií. Čas genetického algoritmu druheho typu selekcie je 1.06 sekundy, riešenie bolo nájdené v 1013 generácie z 6000

generácií. Čas algoritmu zakázaného vyhľadávania je 0.10 sekundy.

Podľa výsledkov tohto konkrétneho testu, Genetický algoritmus v oboch typoch selekcie nájde výsledok lepšie ako zakázaný algoritmus vyhľadávania, ale druhý pracuje konzistentne rýchlejšie, keď prvý pracuje rýchlejšie, iba ak "narazí"na riešenie na mladšej generácii. Ale v iných testoch, kde genetický algoritmus nedokázal nájsť riešenie na mladších generáciách a v algoritme zakázaneho vyhľadávania nebol žiadny cyklus v lokálnom maxime, druhý algoritmus našiel riešenie lepšie ako prvý (a ako vždy, rýchlejšie).

6.3 Čas hľadania riešenia Genetický Algoritmus

Po vykonaní niekoľkých testov na nájdenie riešenia pomocou genetického algoritmu s rôznymi typmi výberu, zozbieraním štatistických údajov z nich a zostrojením nižšie uvedeného grafu možno tvrdiť, že prvý typ výberu sa vyrovná výrazne rýchlejšie ako druhý, hoci druhý typ výberu sa líši od prvého iba prítomnosťou jednej ďalšej mutácie.



	0	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000	2100	2200	2300	2400	2500
Prvy typ selekcie	0,00	0,01	0,07	0,14	0,25	0,29	0,37	0,44	0,52	0,59	0,66	0,73	0,81	0,88	0,95	1,02	1,10	1,17	1,26	1,33	1,41	1,48	1,56	1,63	1,71	1,78
Druhy typ selekcie	0,00	0,10	0,08	0,16	0,27	0,33	0,41	0,49	0,58	0,65	0,73	0,81	0,88	0,96	1,05	1,14	1,25	1,35	1,51	1,63	1,72	1,81	1,89	1,97	2,05	2,14

Obr. 17: Graf z časovymi hodnotami hľadania riešenia rôznymi typami selekcie

6.4 Čas hľadania riešenia Zakazane Prehľadavanie

V prípade algoritmu zakazaneho prehľadavania nemalo zmysel vytvárať graf, keďže pri testoch pre 20 a 30 miest sa výsledky pohybovali od 0,10 sekundy do 0,15 sekundy bez viditeľného vzoru v počte miest.

7 Zaver

Čo sa týka rozdielu medzi dvoma typmi selekcie genetického algoritmu, možno jednoznačne konštatovať, že prvý typ selekcie bol oveľa efektívnejší ako druhý

(a tiež rýchlejší). Čo sa týka rozdielu medzi genetickým algoritmom a zakázaným vyhľadávacím algoritmom, stojí za to povedať, že genetický algoritmus sa ukázal byť stabilnejším v tom zmysle, že skôr či neskôr absolútne nájde riešenie, zatiaľ čo druhý algoritmus (vyhľadávanie tabu) kvôli nešťastnému usporiadaniu miest a nevhodnej veľkosti zoznamu zakázaných štátov sa môžeme dostať do zacyklenia v lokálnom maxime a neposkytnúť nám správny výsledok, aj keď keď poloha miest a veľkosť zoznamu zakázaných štátov zodpovedal, tento algoritmus dal lepší výsledok a desaťkrát rýchlejší ako genetický algoritmus.